present

Programming Experience 2018
- Material Point Method -

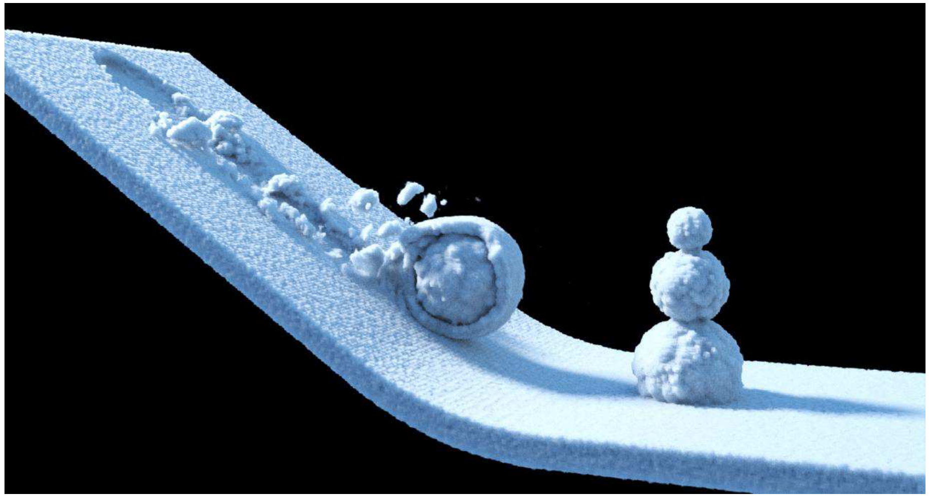
Sebastian Koall

Software Architecture Group
Hasso Plattner Institute
University of Potsdam, Germany

print

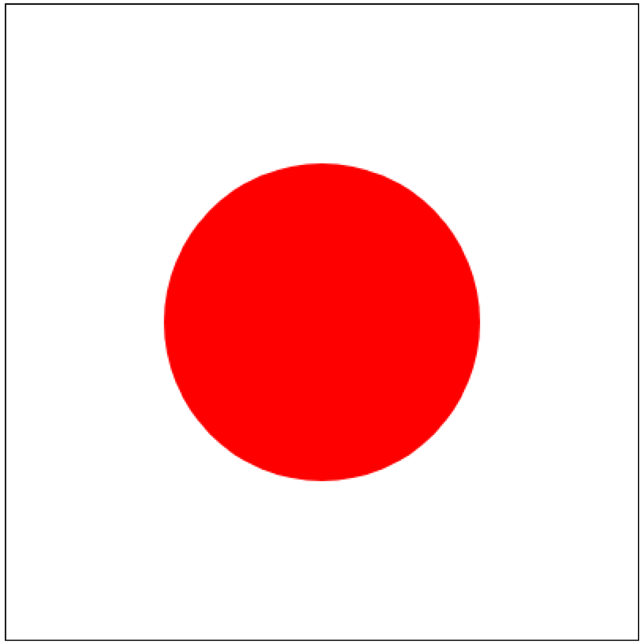
Introduction

- Abbreviation: MPM
- Simulating behavior of: solids, fluids, gas
- Based on Particle-In-Cell Method & Finite Element Method
- Frozen: snow animation
- *Short: tons of formulas*



MPM Overview

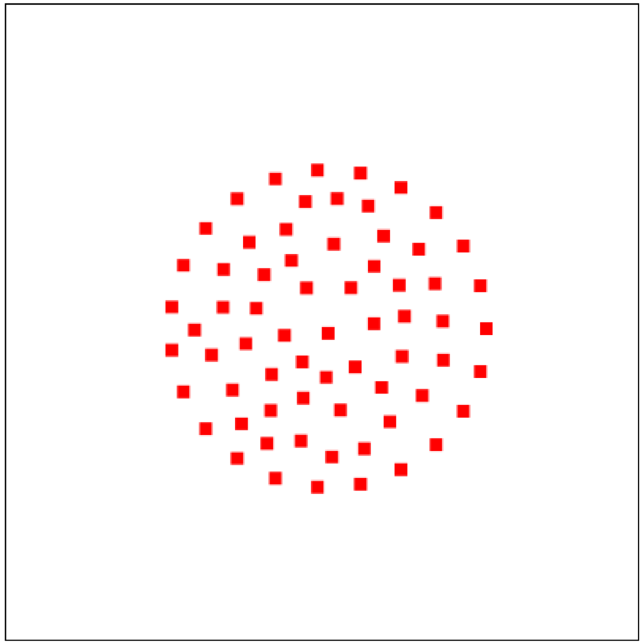
- Continuum body **Ω** discretized into material points **p**
- Located in Euclidean grid with **e** cells (Elements)
- Grid has **n** nodes (2D: $n = (e + 1)^2$)



Previous Next

MPM Overview

- Continuum body **Ω** discretized into material points **p**
- Located in Euclidean grid with **e** cells (Elements)
- Grid has **n** nodes (2D: $n = (e + 1)^2$)

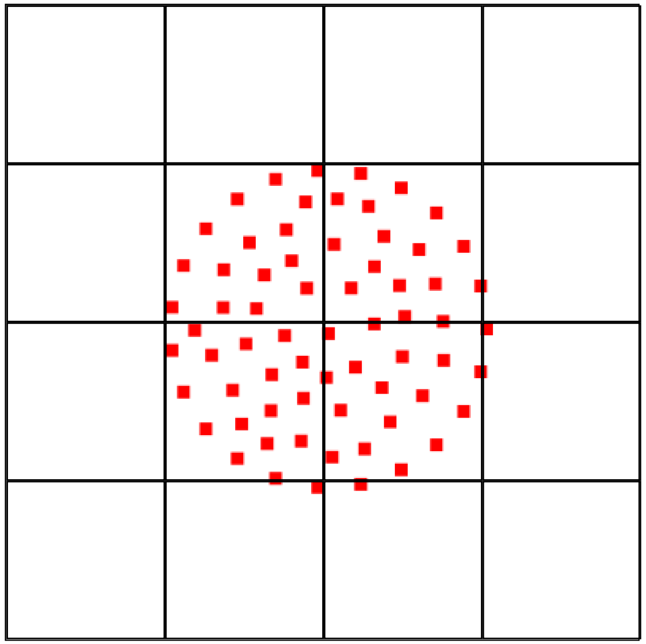


Previous

Next

MPM Overview

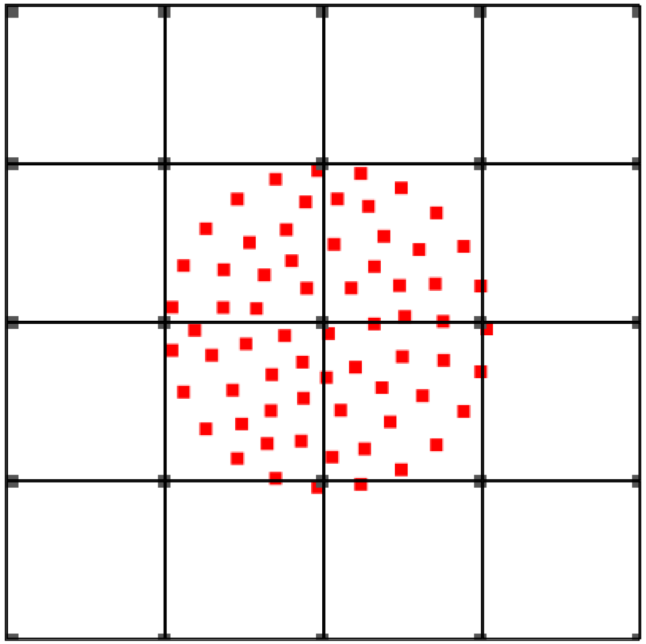
- Continuum body **Ω** discretized into material points **p**
- Located in Euclidean grid with **e** cells (Elements)
- Grid has **n** nodes (2D: $n = (e + 1)^2$)



Previous Next

MPM Overview

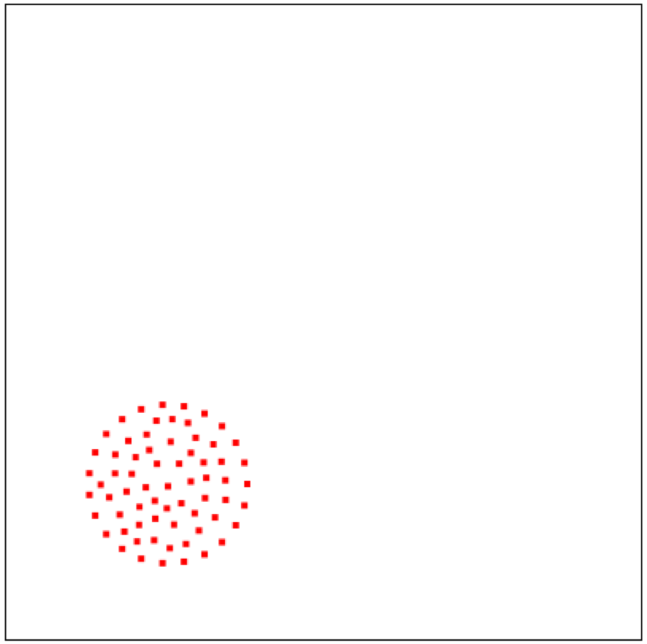
- Continuum body **Ω** discretized into material points **p**
- Located in Euclidean grid with **e** cells (Elements)
- Grid has **n** nodes (2D: $n = (e + 1)^2$)



Previous

Next

Demo 1



Start Animation

Step

Toogle Grid

Reset

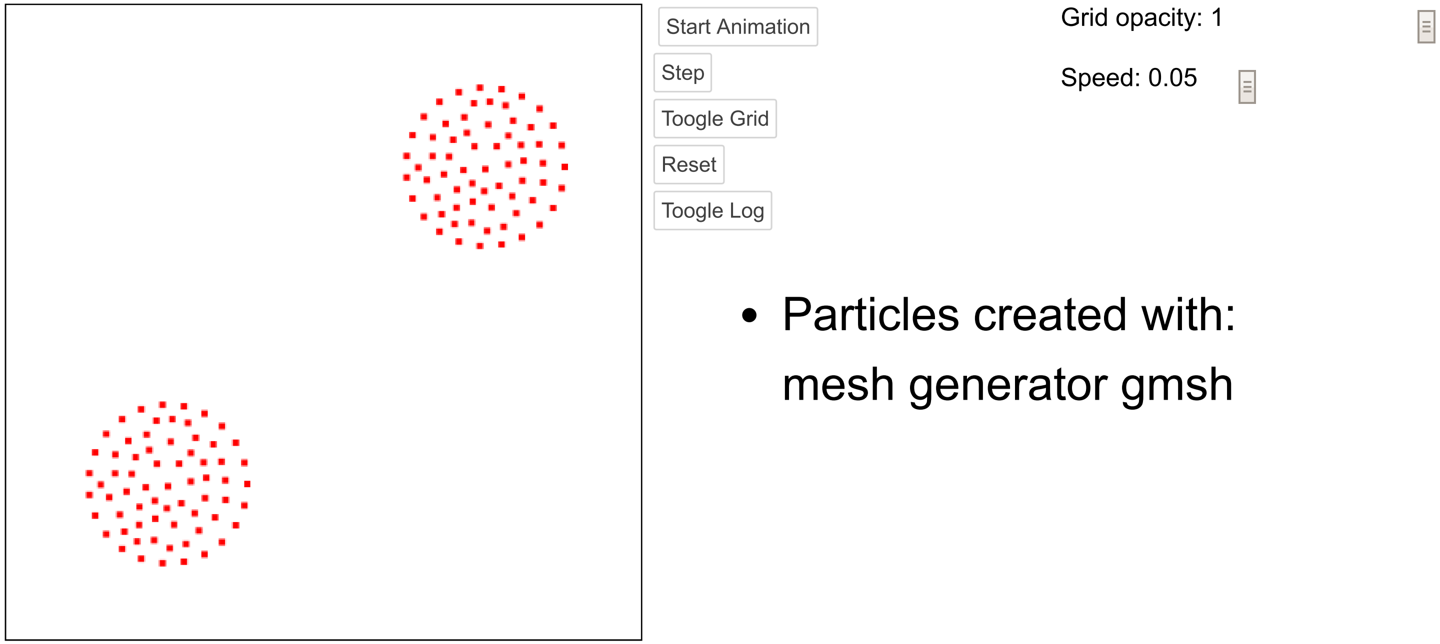
Toogle Log

Grid opacity: 1

Speed: 0.05

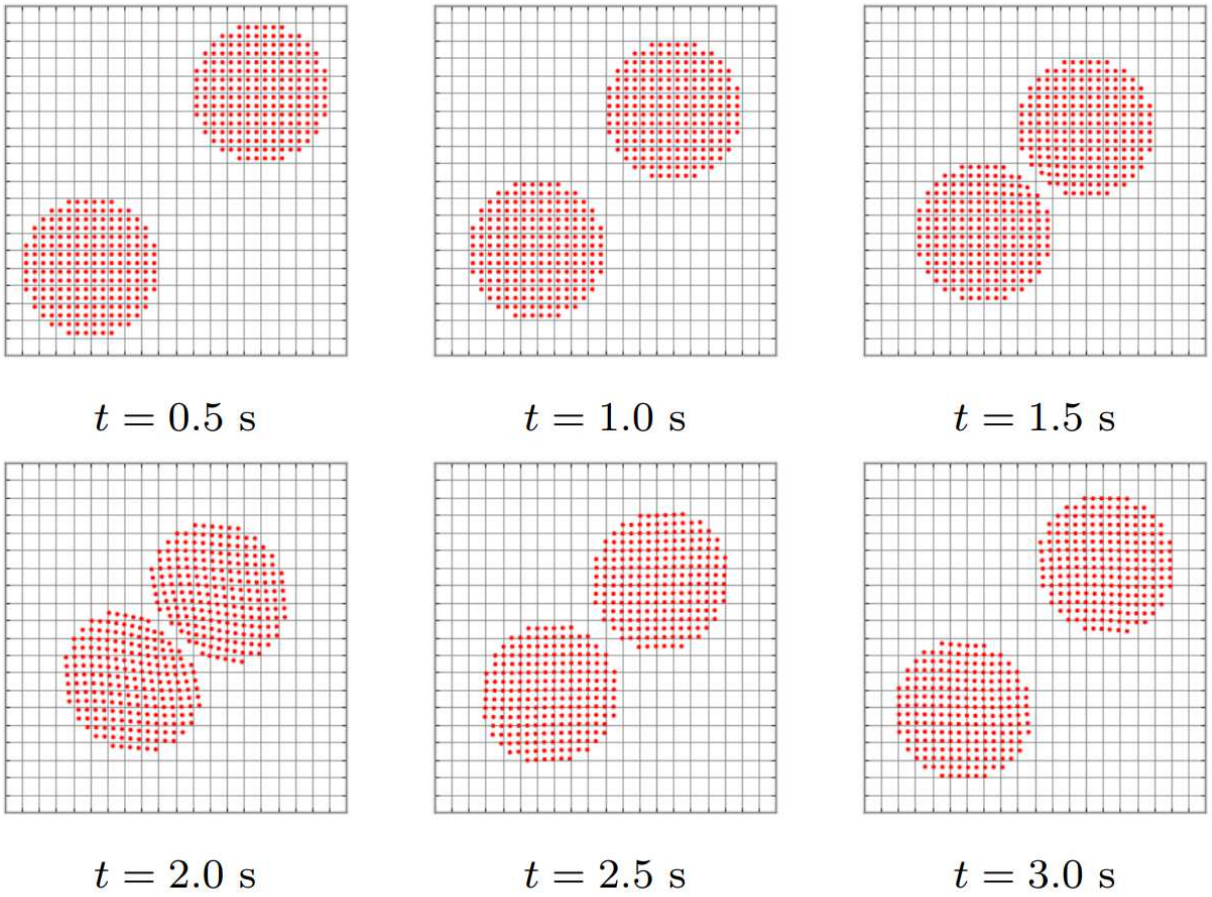
- Particles created with: mesh generator gmsh

Demo 2



- Particles created with: mesh generator gmsh

Correct Result



Related Work

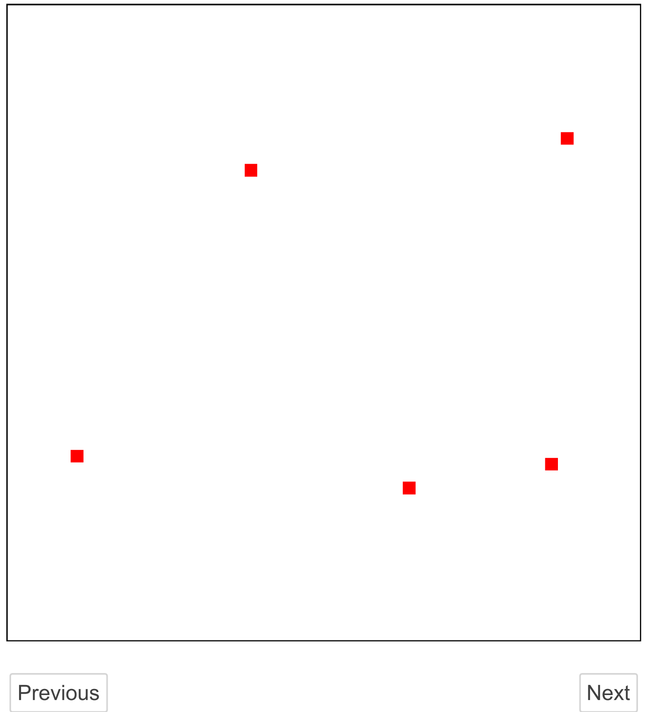
- Original paper:
  A particle method for history-dependent materials
- Basic examples:
  Material point method: basics and applications
- Snow simulation:
  A material point method for snow simulation
- Interpolation:
  Analysis and reduction of quadrature errors in the material point method (MPM)

Processing Loop

- Preparation:
  - Init particle mass, volume and force
  - Init grid
- Loop:
  - Particles to nodes
  - Apply force to momtentum
  - Nodes to particles
  - Reset grid

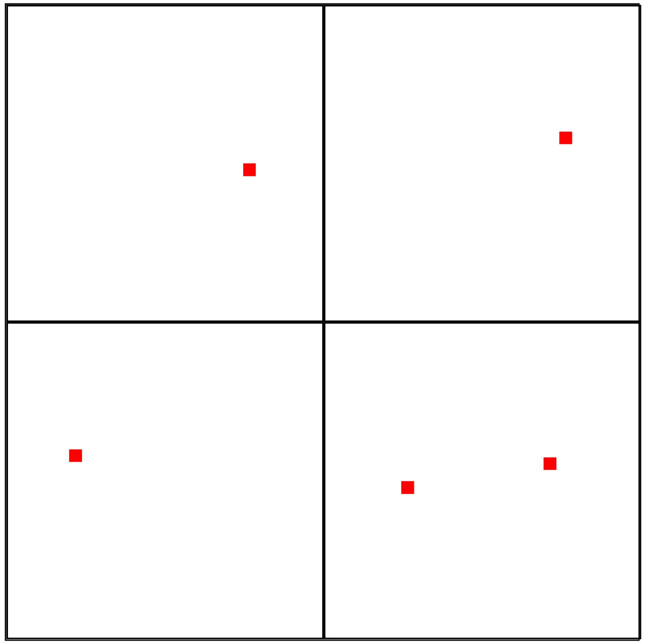
Step: Particles To Nodes

- Maps material points to grid nodes
- Calculate node values:
  - Mass M_i
  - Velocity V_i
  - Force F_i
- Create Lagrange grid



Step: Particles To Nodes

- Maps material points to grid nodes
- Calculate node values:
  - Mass M_i
  - Velocity V_i
  - Force F_i
- Create Lagrange grid

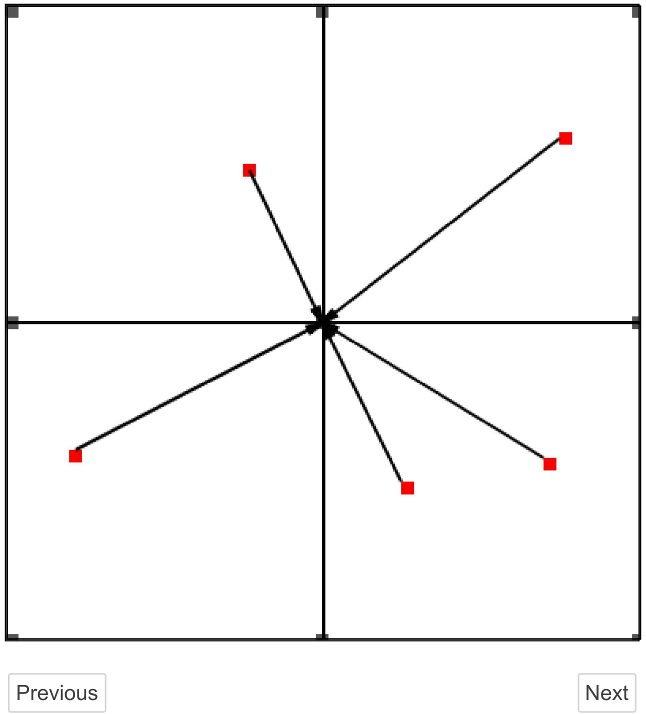


Previous

Next

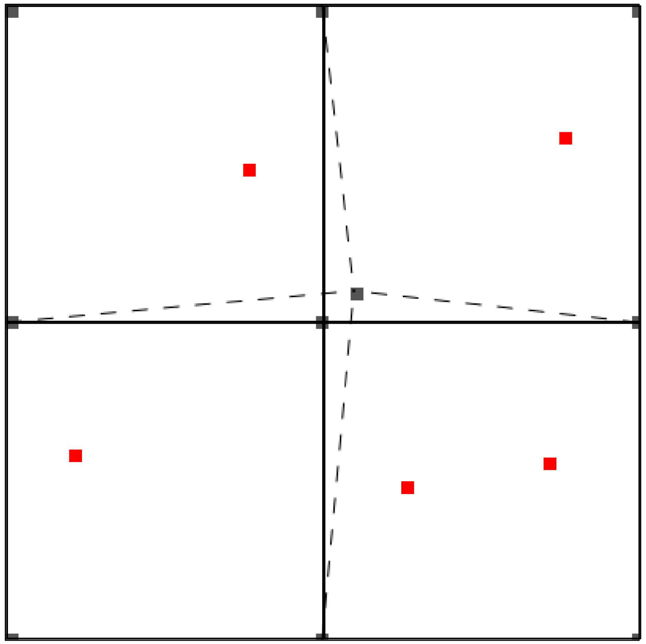
Step: Particles To Nodes

- Maps material points to grid nodes
- Calculate node values:
  - Mass M_i
  - Velocity V_i
  - Force F_i
- Create Lagrange grid



Step: Particles To Nodes

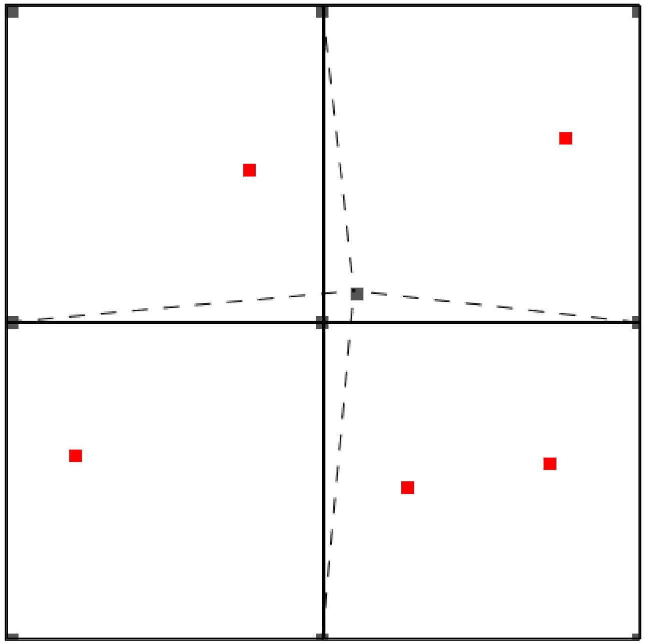
- Maps material points to grid nodes
- Calculate node values:
  - Mass M_i
  - Velocity V_i
  - Force F_i
- Create Lagrange grid



Previous Next

Step: Nodes To Particles

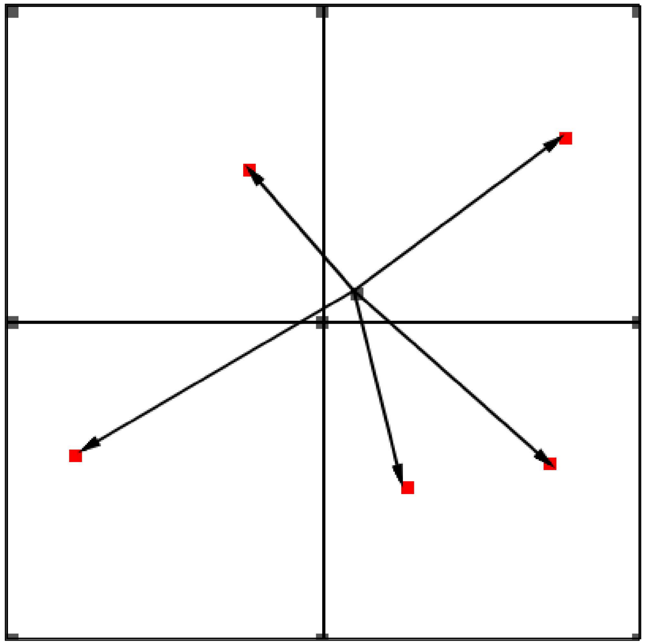
- Map Lagrange grid to particles
- Calculate particle values:
  - Velocity v_p
  - Position x_p
  - Deformation Gradient L_p



Previous Next

Step: Nodes To Particles

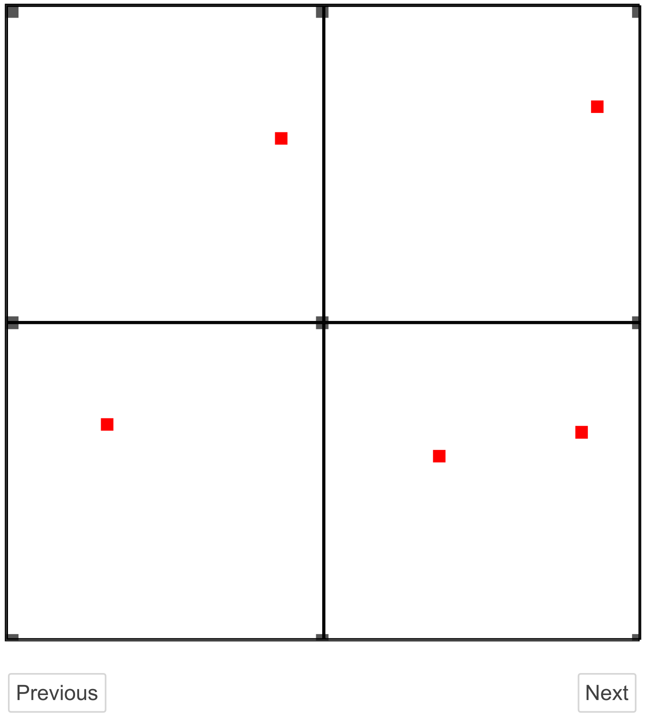
- Map Lagrange grid to particles
- Calculate particle values:
  - Velocity v_p
  - Position x_p
  - Deformation Gradient L_p



Previous Next

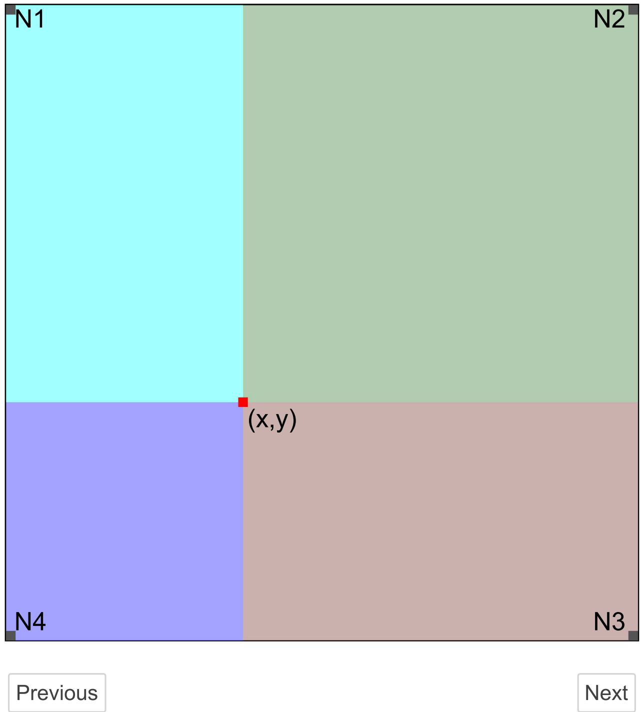
Step: Nodes To Particles

- Map Lagrange grid to particles
- Calculate particle values:
  - Velocity v_p
  - Position x_p
  - Deformation Gradient L_p



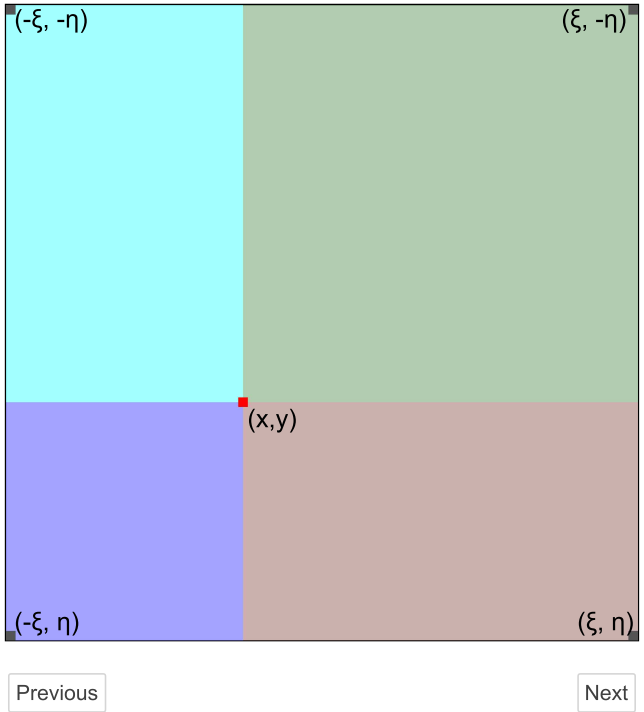
Shape Function

- Degree of influence of nodes on particles and vice versa
- Simple approach: linear interpolation
- Transform into natural coordinates:
    - $\xi = (2 * x - (x_{n1}+x_{n2})) / \Delta x$
    - $\eta = (2 * y - (y_{n1}+y_{n4})) / \Delta y$



Shape Function

- Degree of influence of nodes on particles and vice versa
- Simple approach: linear interpolation
- Transform into natural coordinates:
  - $\xi = (2 * x - (x_{n1}+x_{n2})) / \Delta x$
  - $\eta = (2 * y - (y_{n1}+y_{n4})) / \Delta y$



Shape Function (Linear Interpolation)

- Shape Function:
  - $N_1 = \frac{1}{4} * (1 - \xi) * (1 - \eta)$
  - $N_2 = \frac{1}{4} * (1 + \xi) * (1 - \eta)$
  - $N_3 = \frac{1}{4} * (1 + \xi) * (1 + \eta)$
  - $N_4 = \frac{1}{4} * (1 - \xi) * (1 + \eta)$
- Gradient:
  - $Ndx_1 = \frac{1}{4} * (\eta - 1)$
  - $Ndx_2 = \frac{1}{4} * -(\eta - 1)$
  - $Ndx_3 = \frac{1}{4} * (\eta + 1)$
  - $Ndx_4 = \frac{1}{4} * -(\eta + 1)$
  - $Ndy_1 = \frac{1}{4} * (\xi - 1)$
  - $Ndy_2 = \frac{1}{4} * -(\eta + 1)$
  - $Ndy_3 = \frac{1}{4} * (\xi + 1)$
  - $Ndy_4 = \frac{1}{4} * -(\xi - 1)$

Step: Particles To Nodes - Details

- For each node of every element:
  - Mass: M_i += m_p * N_i
  - Node Momentum: MV_i += N_i * m_p * v_p
  - Force: $F_i = N_i * f_p - Nd_i * \sigma_p * v_p$

Step: Nodes To Particles - Details

- For each node of every element:
  - $v_p = N_i * F_i / M_i * dT$
  - pos_p += $N_i * MV_i / M_i * dT$
  - ΔL += $V_i * Nd' * dT$
- For every element:
  - $L_p = L_p * \Delta L$
  - $V_p = det(L_p) * V_{o\_p}$

Outcomes & Learnings

- MPM is cool (when it works)
- Large number of MPM variables complicates the understanding
- Many MPM papers contain incomplete explanations
- Few code examples only complex systems
- Utilize proper language function when possible (requestAnimationFrame)
- Math.js has an inconvenient syntax